ACEEE Int. J. on Recent Trends in Engineering and Technology, Vol. 10, No. 2, Jan 2014

Reduced Merge_FSM Pattern Matching Algorithm for Network Intrusion Detection

Dr.S.Aruna Mastani
Assistant Professor, JNTUA College of Engineering, Anantapur, Andhra Pradesh, India
Email: aruna_mastani@yahoo.com

***Abstract*— Pattern matching is a significant issue in intrusion detection systems (IDS), as it is required to identify and restrict certain virus patterns by matching them with the patterns present in the database. The performance of an Intrusion Detection System is dependent on two metrics that is throughput and the total number of patterns that can fit on a device. Many hardware approaches are proposed to accelerate pattern matching. Among hardware approaches, memory based architecture has attracted a lot of attention because of its easy reconfiguration and scalability. In memory architecture approach. First, the virus string patterns are compiled to a finite-state machine (FSM) whose output is asserted when any substring of input strings matches the string patterns. The speed of comparisons and memory required to store virus string patterns is evaluated through the number of state transitions made. In this paper, a new pattern-matching algorithm 'Reduced Merge_FSM' is proposed which significantly reduce the memory requirement and provides faster matching, by reducing the state transitions, than that of existing Finite State Machine (FSM) based approaches. This reduction in states results in optimization of memory and also the comparison time. To prove the efficiency of the proposed algorithm, it is compared through experimentation with the existing AC-Algorithm and Merge_FSM.**

***Index Terms*— Aho–Corasick (AC) algorithm, merge_FSM, and pattern matching**

I. INTRODUCTION

The continued discovery of programming errors in network-attached software has driven the introduction of increasingly powerful and devastating attacks. Attacks can cause destruction of data, clogging of network links, and future breaches in security. In order to prevent, or at least mitigate, these attacks, a network administrator can place a firewall or Intrusion Detection System at a network choke-point such as a company's connection to a trunk line. A firewall's function is to filter at the header level; if a connection is attempted to a disallowed port, such as FTP, the connection is refused. This catches many obvious attacks, but in order to detect more subtle attacks, an Intrusion Detection System (IDS) is utilized. The IDS differs from a firewall in that it goes beyond the header, searching the packet contents for virus patterns. Detecting these patterns in the input implies an attack is taking place, or that some disallowed content is being transferred across the network. The detection process is basically a pattern matching of the suspicious strings in the packets with the virus strings stored in the database. Figure1 shows various blocks of a general NIDS. The pattern matching algorithms used for string matching are based on three approaches. First approach is the direct string comparison which stores the strings directly in the database which requires more memory and comparison time. Second one is the comparator method. The drawback of this method is that it requires more area, examples of this are CAM [4] DPCAM [5]. The third approach is the Finite state machine (FSM) approach [2]. In this string patterns are stored in the form of state machines. FSM based algorithms [1]

DOI: 01.IJRTET.10.2.522

reduce the memory size by effectively reducing the number of state transitions. Examples of this are AC-Algorithm [3], Bit-split algorithm [6].The paper concentrates on efficient FSM based algorithms for pattern matching in Network Intrusion Detection System. Section II reviews hardware and software string matching. Section III describes the AC algorithm. Section IV is about the Merge_FSM algorithm and section V concentrates how Merge FSM is modified / extended thus resulting in the proposed method Reduced – Merge_FSM.

II. HARDWARE / SOFTWARE - BASED STRING MATCHING & PACKET INSPECTION

Software-based Intrusion Detection Systems can only support modest throughput. On the other hand, hardware can easily adapt in NIDS application needs, achieving better performance with reasonable cost. The objective of NIDS is to optimizing the Correlator logic for faster compilation to reduce power and area. The basic memory architecture works as follows. First, the (attack) string patterns are compiled to a *finite-state machine* (FSM) whose output is asserted. When any substring of input strings matches the string patterns, then, the corresponding state transition of the FSM is stored in memory. For instance, Figure 2 shows the state transition graph of the FSM to match two string patterns "bcdf" and "pcdg", where all transitions to state 0 are omitted. States 4 and 8 are the final states indicating the matching of string patterns "bcdf" and "pcdg", respectively. Figure 3 presents simple memory architecture to implement the FSM.

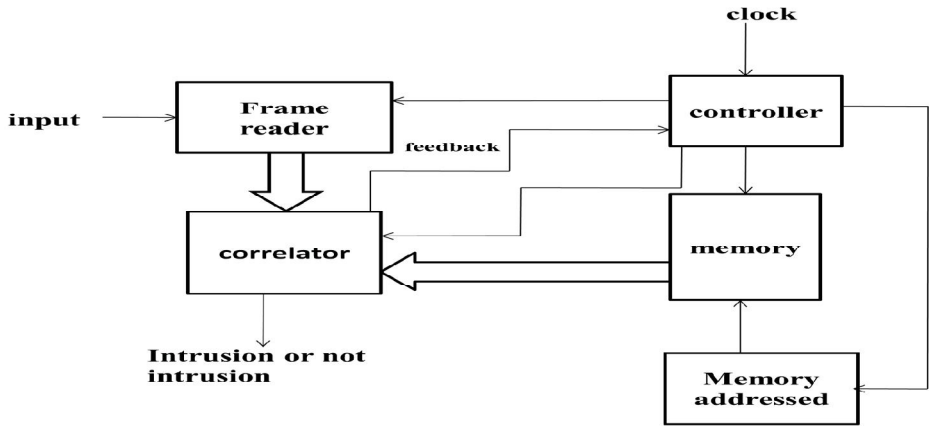

Figure.1 Block diagram of NIDS

In this architecture, the memory address register consists of the current state and input character; the decoder converts the memory address to the corresponding memory location, which stores the next state and the *match vector* information. A "0" in the match vector indicates that no "suspicious" pattern is matched; otherwise the value in the matched vector indicates which pattern is matched. For example in Figure2, suppose the current state is 7 and the input character is g. The decoder will point to the memory location which stores the next state 8 and the match vector 2. Here, the match vector 2 indicates the pattern "pcdg" is matched. Due to the increasing number of attacks, the memory required for implementing the corresponding FSM increases tremendously, thus affecting the performance cost, and power consumption.

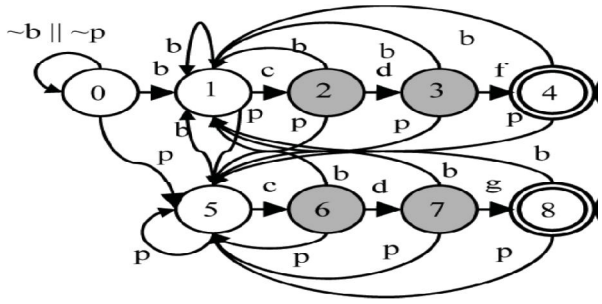

Figure 2. DFA for matching 'bcdf' and 'pcdg'.

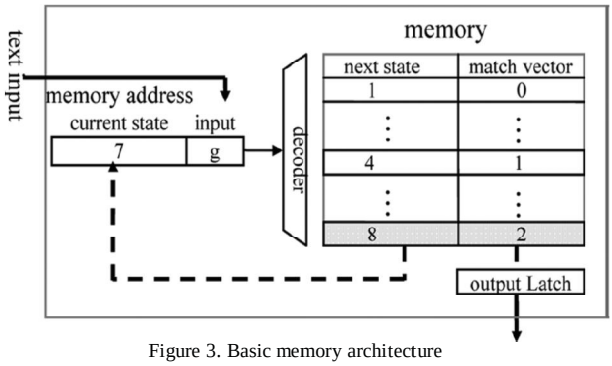

Figure 3. Basic memory architecture

Pattern matching is a significant issue in intrusion detection systems, in this paper, we propose an algorithm 'Reduced Merge_FSM' to address the challenge of payload pattern matching in intrusion detection systems. The performance of the method proposed is compared with the existing efficient pattern matching techniques to analyze packet payloads at multi gigabit rates and detect hazardous contents. The techniques used are **t**he first is, decoded partial CAM [4] (DPCAM), it pre decodes incoming characters, aligns the decoded data, and performs logical AND on them to produce the match signal for each pattern. The second is the Aho–Corasick (AC) algorithm [3] based FSM. The third technique is the Merge_FSM (memory-efficient pattern-matching algorithm**).**

III. 'AC' ALGORITHM

Consider the same example as in Figures 2 and 3, Figure 4 shows the state transition diagram derived from the AC algorithm where the solid lines represent the *valid* transitions while the dotted lines represent a new type of state transition called the *failure* transitions. The failure transition is explained as follows. Given a current state and an input character, the AC machine first checks whether there is a valid transition for the input character; otherwise, the machine jumps to the next state where the failure transition points. Then, the machine recursively considers the same input character until the character causes a valid transition. The AC machine takes a failure transition back to state 0. Then in the next cycle, the AC machine reconsiders the same input character in state 0 and finds a valid transition to state 5. This example shows that an AC machine may take more than one cycle to process an input character.

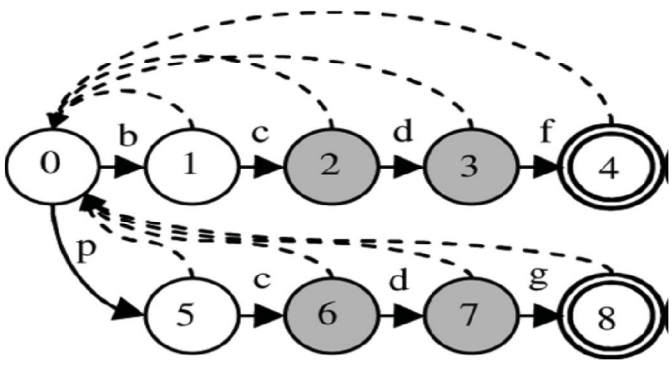

Figure 4. State diagram of an AC machine

In Fig. 6, the double-circled nodes indicate the final states of patterns. In Fig. 3, state 4, the final state of the first string pattern "bcdf", stores the match vector $\{P_2P_1\}=\{01\}$and state 8, the final state of the second string pattern "pcdg", stores the match vector of$\{P_2P_1\}=\{10\}$. Except the final states, the other states store the match vector $\{P_2P_1\} = \{00\}$ to simply express those states are not final states. In this AC algorithm the drawback is that the common substrings are not merged to reduce the number of states that leads to more state transitions. This has been overcome in the other algorithm called Merge_FSM algorithm which is discussed below.

IV. MERG_FSM ALGORITHM

Continuing with the example considered above in AC algorithm ,state 26 represents two different states (state 2 and state 6) and state 37 represents two different states (state 3 and state 7). It is shown that directly merging similar states leads to an erroneous state machine, to have a correct result, when state 26 is reached, there is a need of a mechanism to understand in the original AC state machine whether it is state 2 or state 6. Similarly, when state 37 is reached, it is necessary to know that in the original AC state machine whether it is state 3 or state 7. In this example, state 2 or state 6 can be differentiated, if it is possible to memorize the precedent state of state 26. If the precedent state of state 26 is state 1, then it can be known that the original AC state machine, is in state 2. On the other hand, if the precedent state of state 26 is state 5, the original AC state machine, is in state 6. From this it is inferred that if it is possible to memorize the precedent state before entering the merged states, then all merged states can be differentiated. in the following sections, we discuss the retaining of the precedent path vector during the state traversal in the Merge FSM, along with the modification that leads to the proposed method Reduced Merge_FSM.

V. REDUCED MERGE_FSM

In the Merge FSM though the common substrings in all the string patterns are taken there is a still a scope to reduce the state transition time by merging common substrings which may be common even for few patterns, This is the basic principle behind the proposed method 'Reduced Merge FSM', that leads to further reduction in state transitions and fast pattern matching. For example Let us consider three virus string patterns 'abcdef', 'kbgdyh', 'wxpdem'. According to merge_FSM technique the common substring among the three substrings is *d* but if we observe the 'abcdef', 'kbgdyh' strings there is a common substring *b.* Similarly if we observe the 'kbgdyh', 'wxpdem' strings there is a common substring *g.* By modifying the merge FSM technique and performing further merging it is possible to reduce the number of state transitions.

VI. ANALYSIS AND COMPARISON OF AC ALGORITHM, MERG_FSM, REDUCED MERG_FSM

Consider three virus string patterns 'abcdef', 'kbgdyh', 'wxpdem'. The AC state machine for the above three patterns is constructed as follows. Initially if the state machine is in zero state, then based on the current input it moves to the next state corresponding to that input. By observing the state machine in Fig.5, the final state of the first string pattern 'abcdef', stores the match vector $\{P_3P_2P_1\}=\{001\}$ and state6, similarly the final state of the second string pattern 'kbgdyh', stores the match vector $\{P_3P_2P_1\}=\{010\}$ and state12, lastly it is also observed that the final state of the third string pattern 'wxpdem', stores the match vector of $\{P_3P_2P_1\}=\{100\}$ and state18. From the state machine shown in fig.5, it is interpreted that except the final states, the other states store the match vector $\{P_3P_2P_1\} = \{000\}$.

The construction of a state traversal machine consists of: 1) The construction of valid transition, failure transition, pathVec, and ifFinal functions 2) merging pseudo-equivalent states. In the first step, the states and valid transitions are created , and then, the failure transitions are created. The construction of pathVec and ifFinal begins in the first step and completes in the second step.

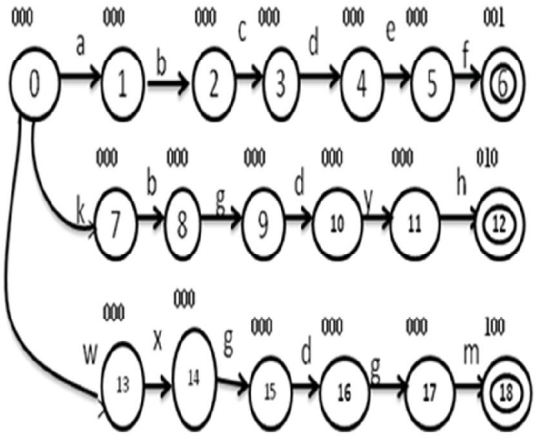

Figure 5. AC state machine for the three string patterns

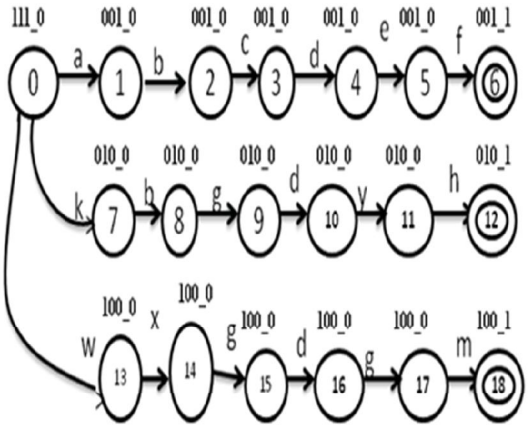

Figure 6. Direct graph showing Construction of pathVec and ifFinal

For example, consider the three patterns ‘abcdef’, ‘kbgdyh’, ‘wxpdem’. Adding the first pattern ‘abcdef’ to the graph, the path from state 0 to state 6 matches the first pattern ‘abcdef’. Therefore, the pathVec of all states on the path is set to $\{p_3p_2p_1\} = \{001\}$, and the ifFinal of state 6 is set to 1 to notify it as final state where the path terminates. Adding the second pattern ‘kbgdyh’ into the graph, the pathVec of other states, {state 7, state 8, state 9, state 10, state 11 and state 12} on the path is set to $\{p_3p_2p_1\}=\{010\}$ further, the ifFinal of state 12 is set to 1 to indicate the final state for the second pattern. Similarly, when the third pattern ‘wxpdem’ is added to the graph the pathVec of other states {state 13, state 14, state 15, state 16, state 17, and state 18} on the path is set to$\{p_3p_2p_1\}=\{100\}$. The ifFinal of state 18 is set to 1 to indicate the final state of the third pattern.

We can find that states 4, 10, and 16 are pseudo-equivalent states because they have identical input transitions, identical failure transitions to state 0 and identical ifFinal 0. Those pseudo-equivalent states are merged to state 4_10_16. The pathVec of state 4_10_16 is modified to be $\{p_3p_2p_1\} = \{001\} \parallel \{010\} \parallel \{100\}=\{111\}$. Fig.7 shows the final state diagram of Merge FSM. Comparing it with the original AC state machine in Fig.5, we find that two states are eliminated.

From the Fig.6 States 2, 8 and states 5, 17 are a pseudo-equivalent state that is they have same input and different output. By merging the states 2,8 and states 5,17 it is possible to reduce the number of state transitions. Reduced merge_FSM extracts and merges the pseudo-equivalent states. Notice that merging pseudo-equivalent states includes merging the failure transitions and performing the union on the pathVec of the merged states. We can find that states 2 and 8 are pseudo-equivalent states because they have identical input transitions, identical failure transitions to state 0 and identical ifFinal 0. Similarly, states 5 and 17 are pseudo-equivalent states. Those pseudo-equivalent states are merged to states 2_8 and 5_17 respectively. The pathVec of state 2_8 is modified to be $\{p_3p_2p_1\} = \{001\} \parallel \{010\} = \{011\}$ by performing the union on the pathVec of state 3 and state 8. Similarly, the pathVec of state 5_17 is also modified to be $\{p_3p_2p_1\} = \{010\} \parallel \{100\} = \{110\}$. Fig.8 shows the final state diagram of reduced merge_FSM.

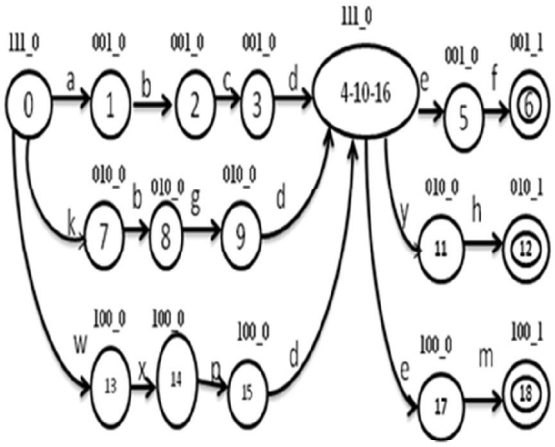

Figure 7. State diagram of Merge FSM

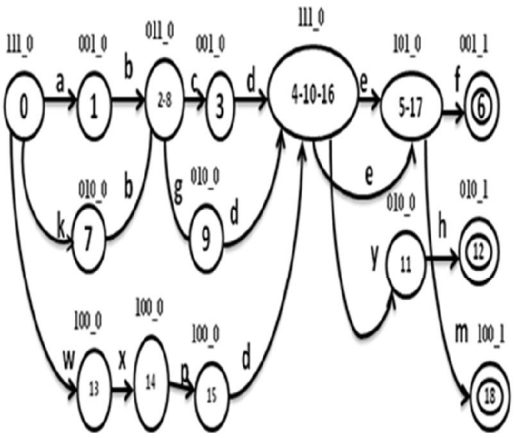

Figure 8. State diagram of Reduced Merge FSM

VII. EXPERIMENTAL RESULTS

From the detail description made through examples it is inferred that by considering three string patterns containing 6 characters each ‘abcdef’, ‘kbgdyh’, ‘wxpdem’. Through Merge_FSM technique the number of transitions is 19. Similarly after applying reduced merge_FSM technique the number of transitions is 16.The results are compared with the previous AC-Algorithm where the number of transitions is 21. In the above three strings the common substring is ‘*d*’. From the two strings ‘abcdef’, ‘kbgdyh’ the common substring is *b*. Similarly in case of ‘abcdef’ ‘wxpdem’ the common substring is ‘*e*’. In case of the above proposed algorithm it takes one common memory location to store the substring state information. If the same experiment is repeated for ‘*x*’ number of times the number of transitions remains same and the time taken to compare the two string patterns is same. The results are tabulated in table.1 , from this it is observed that the state transitions are least for the proposed method, Thus provides significant reduction in the memory requirement and hence faster matching,

Consider *N* number of string patterns in which *K* number of string patterns contains the common substring patterns of length *n* .AC-Algorithm takes *K* separate memory locations of length *n* to store each substring state information. . In case of the above proposed algorithm it takes one common memory location of length *n* to store the substring state information. The memory is reduced from *K*n* to *n* hence the memory saved is *(K-1)*n.* Comparison with respect to number of state transitions.

TABLE I: COMPARISON OF AC ALGORITHM, MERGE_ FSM AND REDUCED MERGE_FSM

Method	Patterns	Characters	Number of transitions
AC algorithm	3	16	21
Merge_FSM	3	16	19
Reduced Merge FSM	3	16	16

VIII. CONCLUSIONS

The proposed algorithm reduced merge_ FSM significantly reduce the number of states and state transitions by merging pseudo-equivalent states yet e maintaining correct string matching. In addition, the this algorithm is complementary to other pattern matching algorithms and enhance reductions in memory needs leading to increase in speed of comparison. The future work intends is the Hardware development for reduced merge_ FSM

REFERENCES

[1] M. Aldwairi, T. Conte, and P. Franzon, "Configurable string matching hardware for speeding up intrusion detection," Proc. Acm Sigarch Comput. Arch. News, vol. 33, no. 1, pp. 99–107, 2005.

[2] H. Lu, K. Zheng, B. Liu, X. Zhang, and Y. Liu, "A memory-efficient parallel string matching architecture for high-speed intrusion detection," IEEE J. Sel. Areas Commun., vol. 24, no. 10, pp. 1793–1804,Oct. 2006.

[3] A. V. Aho and M. J. Corasick, "Efficient string matching: An AID to bibliographic search," Commun. ACM, vol. 18, no. 6, pp. 333–340, 1975.

[4] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. H. Granidt, "Towards gigabit rate network intrusion detection," in Proc.the Eleventh Annual ACM/SIGDA International Conference on Field- Programmable Logic and Applications (FPL '03), 2002, pp. 404–413.

[5] C. R. Clark and D. E. Schimmel, "Scalable pattern matching on high speed networks," in Proc. 12th Ann. IEEE Symp. Field Program. Custom Comput. Mach. (FCCM), 2004, pp. 249–257.

[6] Bin Liu,and Yunhao Liu, "A Memory-Efficient Parallel String Matching Architecture for High-Speed Intrusion Detection", in Proc. VOL. 24, NO. 10, October 2006